www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Brief overview of Machine Learning towards Benchmarking

Josep Ll. Berral-García
Researcher & Data Scientist

Index

Machine Learning Algorithms and Methods for Big Data

1. A very brief course on Machine Learning
2. Some algorithms by kind
3. Performance concerns for ML
4. Current trends and State of the Art
5. Code examples and Cases of Use

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

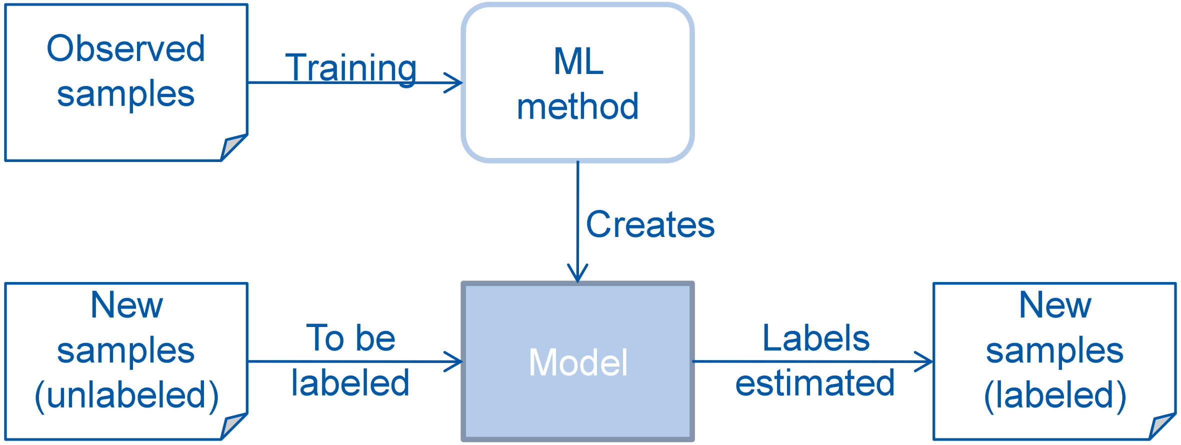
EXCELENCIA SEVERO OCHOA

A VERY BRIEF COURSE ON MACHINE LEARNING

What is Machine Learning

- Machine Learning:
  - Data mining science and methods…
  - …In charge of learn a system (modeling) automatically…
  - …From some of its observations

- Modeling and Prediction



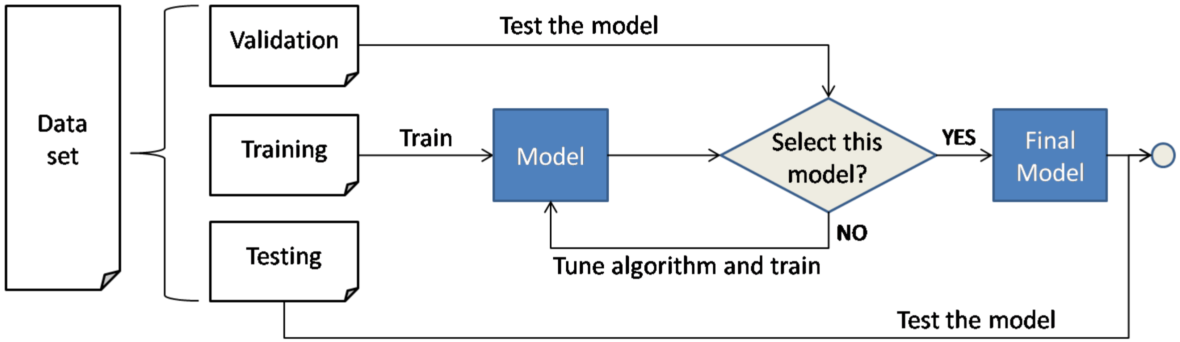
  - Usually the model learns "f(inputs) → labels"
  - "f(inputs)" explains the observed system

Usual methodology

1. Collect samples from the system
   – Label them a-posteriori (if possible)

2. Select which data features are used

3. Split the data for training and testing
   - Split the data-set in training, validation and testing sets
   - Alternatively, use “cross-validation”, in case of few data available
   – Never test your model with the training data!



Capabilities of learning

- What machine learning can do:
  - Find the function or rules that define labels in data
  - Estimate values (regression) or classes/categories (classification)

  - Estimate unseen inputs for a given model
  - Find labels from unlabeled data, and label the training and test data
  - Recommend actions from past seen action results

  - Show how a system works by printing its learned function

- What machine learning can not do (but can help):
  - Solve GI, NP or harder problems
  - Make decisions

Methodology

- Most part of the process falls into Data Preparation:
  - Slice data, Sample data, Aggregate data, Discard data, Flatten data
- Most of learning processes just read data once, then perform an aggregate operation or iterate over data until converging
- Most of prediction processes are just
  - Computing the new input into the model (easy)
  - Compare the input with the model (hard)
- Most predictions require Data to be Post-Processed into readable information

Learning

```
## Machine Learning Typical Program

data <- read.data(source);                                            # Obtain Data
proc.data <- treat.data(data);                                        # Pre-process data
model <- learning.process(proc.data);                                 # Here learning occours

new.proc.data <- treat.data(new.data);                                # Pre-process new data
predicted.values <- prediction(model, new.proc.data);                 # Here prediction occours
displayable.data <- preprint.data(predicted.values, new.proc.data);   # Post-process data
show.data(displayable.data);                                          # ...to be displayed
```

Prediction

ML Programmatically (Part 1)

Languages and Libraries

- R-cran
  - R → Functional/Imperative Programming Language
  - Community libraries for lots of algorithms
  - Parallelization of the "apply" operator (vectorization)
    - "parallel", "snow" / "snowfall",...
    - RHadoop, SparkR + SparkSQL

- Python: Sci-Kit and SK-Learn
  - Machine Learning Library for Python
  - Compatibility towards parallelism on CPUs, GPUs and Clusters
    - Theano, Caffe, Lassagne …

- Java: Weka and MOA
  - Java libraries with ML and Stream Learning algorithms
  - Documented in "I.Witten, E.Frank, M.A.Hall, *Data Mining 3rd Ed.* 2011"

ML Programmatically (Part 2)

- Tools and Frameworks
  - Google's TensorFlow
    - Framework oriented towards Machine (Deep) Learning
    - Programmable in Python
  - Visualization: ElasticSearch
    - Framework to store and visualize data, managed by JSON queries.

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

SOME ALGORITHMS BY KIND

Some algorithms by kind (I)

- **Supervised learning:** Classification and Regression
  - Regression algorithms: Linear/polynomial/logistic regression
    - Learning requires to iterate over data to find weights for each input. Predict implies to perform parallel products and a sum.
  - Decision trees: CART, Recursive Partition, ID3, C4.5, M5P
    - Learning requires recursive algorithms separating data, to find adequate partitions. Prediction implies navigate a tree.
  - Spatial models: Nearest Neighbors
    - Learning puts sample data into memory. Prediction requires to compare inputs with all memorized data.
  - Bayesian models: Naïve-Bayes, Bayesian Networks.
    - Learning implies counting every element from inputs. Prediction becomes to apply Bayes with some counters.
  - Artificial Neural Networks: Neural Networks, Support Vector Machines
    - ANNs require an iterative intensive process over data. Prediction becomes a matrices multiplication.

Some algorithms by kind (II)

- Unsupervised learning: Clustering or Knowledge Discovery
  - Clustering: k-Means, DBSCAN
    - Modeling through (unbounded) iterative processes.
  - Neural Networks: Deep-belief Neural Networks, Boltzmann Machines
    - DBNs require heavy iterative learning processes.

- Reinforcement learning: Model updates from Feedback
  - Look-up tables, Neuro-dynamics
    - Learning through execution → Trial/Error (State-Action counting). Usually requires memory, and undefined amount of retries to learn. ANNs are used to reduce memory.

- Data-Streams: Model learns continuously
  - Stream sketches
    - IO-bound process. Data is treated once (or none). Learning consists in keeping a limited number of counters or samples in memory.

Some algorithms by kind (III)

- **Data Mining:** Finding patterns and relations
  - Collaborative Filtering
    - Memorizing inputs to determine values missing elements by comparison
  - Association Rules
    - A-priori algorithms
    - Counting elements from all inputs, and their combinations
  - Knowledge Discovery
    - Find relevant patterns, using some of the previous techniques, often in an exhaustive way.

- **Reunion of algorithms**
  - Bagging, Boosting, joining freely different algorithms…

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

PERFORMANCE CONCERNS

Performance Concerns (I)

- Algorithm Precision/Recall/Accuracy...
  - Measures of “how good the algorithm learns and predicts”

- Resource usage:
  1. Time and memory for learning process
     - Relation P/R/A against time spent and memory used
  2. Time and memory for predicting new data
     - How much memory requires a model
     - How much time takes to compute a single prediction
  3. Reusing the model
     - Have I to retrain the model each time
     - How much storage takes my model
  4. Visiting the data
     - How many times the data must be visited to create a model

Performance Concerns (II)

- Examples:
  1. Time and memory for learning process
     - k-NN dumps data to memory (↑ memory, ↓ time)
     - N-Nets iterate over data until converge on a matrix (↓ memory, ↑ time)
     - Accuracy on N-Nets depend on iterations [user decides] (time ~ accuracy)
  2. Time and memory for predicting new data
     - LinReg predicts through a multiplication and a sum (↓ time)
     - k-NN must check the input against all memorized data (↑ time)
  3. Reusing the model
     - k-NN updates by memorizing new data (↓ time)
     - Decision Trees require rebuilding them for new data (restart the process)
  4. Visiting the data
     - Naïve-Bayes visits each element in data once
     - N-Nets visit each element *K* times or more (*K* ~ thousands or more)

ML Scenarios (I)

- Details:
  - ML is not (usually) a real time process by itself
    - Except Stream Learning, so it is very resource-bounded
  - The same ML process, with same parameters and different resources available, should produce the same result but in different times
    - Some ML processes have stochastic functions inside, so not the same exact, but similar
  - Once a model is learned, it may be used for processing lots of data
    1. We want to know how to classify/predict, and new examples come slowly
       - We focus on learning time. All attention comes to learning with accuracy
    2. We want an acceptable model to process lots of data
       - We focus on prediction performance. All attention comes to processing new data

ML Scenarios (II)

- The kind of input data affects the ML process:
  - Numeric input:
    - Algorithms can operate numbers easily
  - String inputs:
    - Strings must be parsed to operate and compare
    - This brings slowness to the data processing
  - Label inputs:
    - We have known limited strings, that can be numbered, so they become easy to treat

- Sometimes data must be all in memory:
  - For some algorithms, data must be upload all or partially to memory
    - E.g. N-Nets use batches of data, repeatedly uploaded and used

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

CURRENT TRENDS AND STATE OF THE ART

Current Trends and State of the Art

- On top of the hype
  - Deep Learning and Neural Networks
    - Google's TensorFlow, Theano (GPUs), …
  - Spark platforms
    - Mllib (in brief SparkML), SparkR, SparkSQL, …
  - SQL + R
    - In-Database Analytics:
      - User Defined Functions
      - Embedded R procedures into SQL execution trees.
    - Current commercial providers:
      - MS SQLServer 2016, IBM PDA, Cisco Parstream…

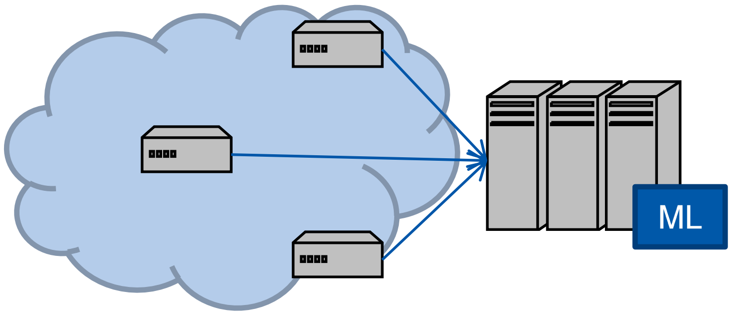
Consolidated methods

- In Academy:
  - Tools
    - R libraries (with parallelism packages)
    - Python SciKit-Learn
    - Scala and Julia
  - Classical learning and ANNs
    - Python+Theano, TensorFlow

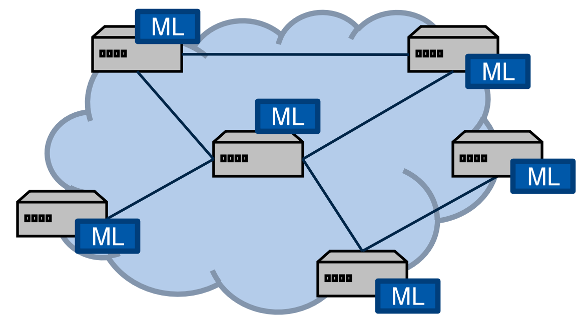
- In Industry:
  - Recommendation modules → Association Rules, ANNs, Collaborative Filtering …
  - User modeling → ANNs, Bayesian Methods, Clustering, …
  - Information retrieval → Data Archeology (Data Mining + Knowledge Discovery), …
  - Fraud Detection → ANNs and Collab. Filtering, …

Scenarios for Modeling and Prediction

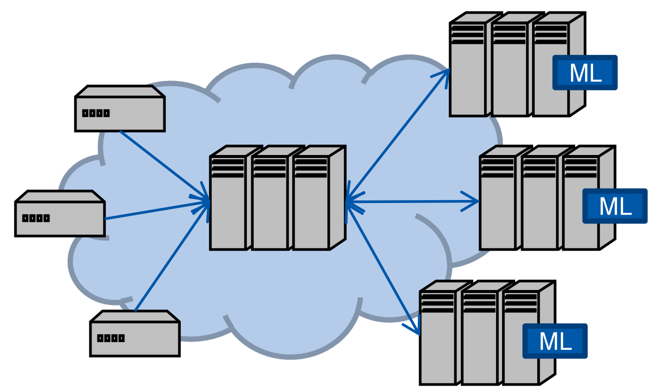
- Static scenarios
  - We collect all the data in one place
  - We train a model
  - We use the model



- Distributed Scenarios
  - Each place collects its data
  - Each place trains its model
  - Each place uses its model



- Map-Reduce Scenarios
  - Data is distributed on places
  - Each place trains part of the model
  - The complete model is collected and used



- Lots of other scenarios

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

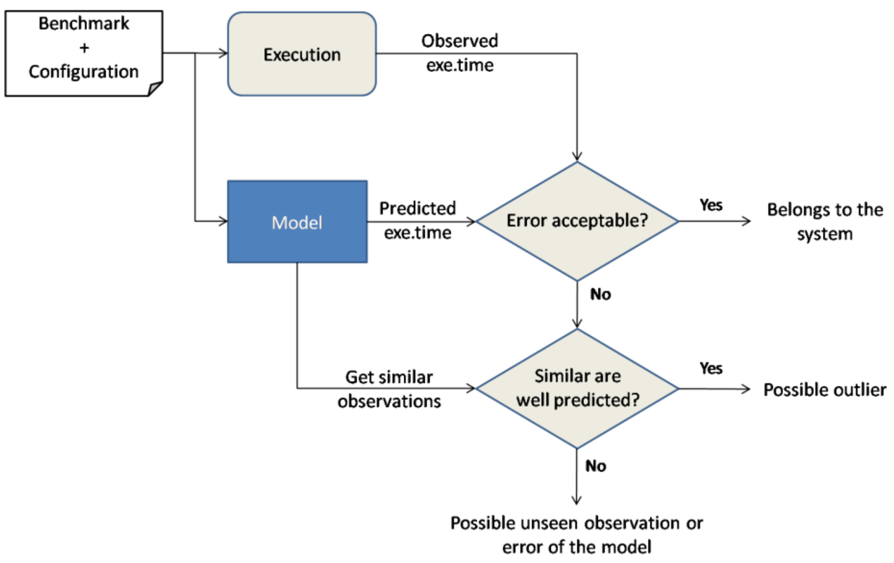
EXCELENCIA SEVERO OCHOA

CODE EXAMPLES AND CASES OF USE

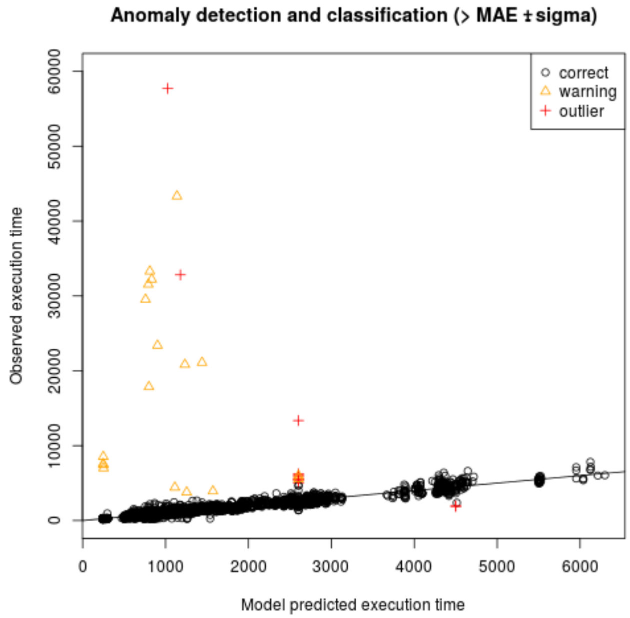
ALOJA-ML – Anomaly Detection

- Anomaly Detection
  - Model-based detection procedure
  - Pass executions through the model
  - Executions not fitting the model are considered “out of the system”

Anomaly detection procedure:



Testing ALOJA Data-set:

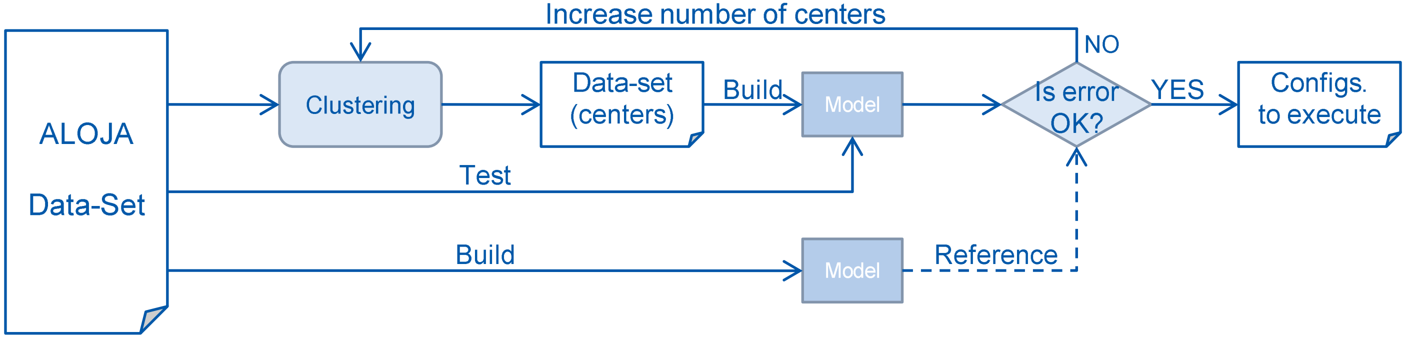


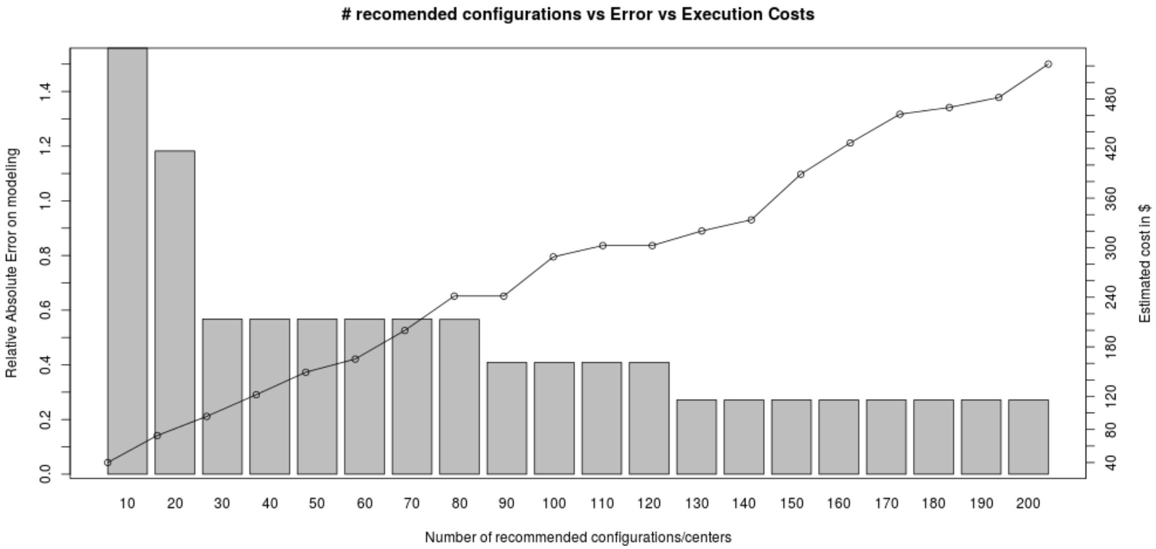
* “ALOJA-ML: A Framework for Automating Characterization and Knowledge Discovery in Hadoop Deployments”. KDD 2015

ALOJA-ML – Guided Benchmarking

Guided Benchmarking:

- Best subset of configurations for modeling Hadoop deployments
- Clustering to reduce the realized executions into the significant ones
- Recommend for future usage the reduced set of executions





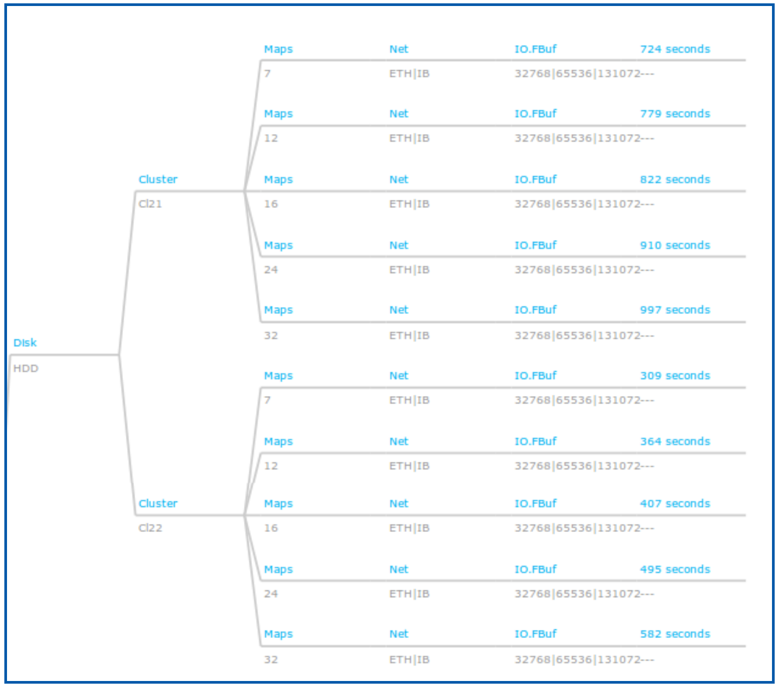
* “ALOJA-ML: A Framework for Automating Characterization and Knowledge Discovery in Hadoop Deployments”. KDD 2015

ALOJA-ML – Discrimination and Representation of Features

- Tools for treating and visualizing data, observed and predicted
- Find best expected executions
  - Use models to predict search sub-spaces
- Discrimination of Features
  - Create a ranking of features, find ways to display them…
  - Possible discrimination by information gain, ordered splits, or other relevant indicators

Benchmark	Net	Disk	Maps	IO.FBuf	Cluster	Prediction
terasort	ETH	HDD	16	32768	Cl21	822.4670000
terasort	ETH	HDD	24	32768	Cl21	909.9230000
terasort	ETH	HDD	7	32768	Cl21	724.0790000
terasort	ETH	HDD	12	32768	Cl21	778.7390000
terasort	ETH	HDD	32	32768	Cl21	997.3790000
terasort	ETH	HDD	16	65536	Cl21	822.4670000
terasort	ETH	HDD	24	65536	Cl21	909.9230000
terasort	ETH	HDD	7	65536	Cl21	724.0790000
terasort	ETH	HDD	12	65536	Cl21	778.7390000
terasort	ETH	HDD	32	65536	Cl21	997.3790000

. . .



* "ALOJA: A Framework for Benchmarking and Predictive Analytics in Big Data Deployments". IEEE TETC 2015

Apache Spark Examples

The classical Word-Count

```
val conf = new SparkConf()
val sc = new SparkContext(conf)

val data = sc.textFile("file")                        // Get file in HDFS to parse
val tokens = data.flatMap(_.split(" "))               // Split content by spaces

val wordFreq = tokens.map((_, 1)).reduceByKey(_+_)     // Map each word, then add count when
                                                         reducing

wordFreq.sortBy(s => -s._2).map(x => (x._2, x._1)).top(10) // Distributed sort, get top 10
```

A MLlib example

```
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

val data = sc.textFile("data/mllib/kmeans_data.txt")
val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()

val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(parsedData, numClusters, numIterations)

clusters.save(sc, "myModelPath")
```

* Examples from Wikipedia and the Apache Spark web-site

SparkR + SparkSQL

```
Sys.setenv(SPARK_HOME='/usr/hdp/2.4.2.0-258/spark');
.libPaths(c(file.path(Sys.getenv('SPARK_HOME'), 'R', 'lib'), .libPaths()));
library(SparkR);


sc <- sparkR.init();
sqlContext <- sparkRSQL.init(sc);


# Example 1
housing_sub <- read.parquet(sqlContext, "hvalp1000.parquet");
aggreg <- SparkR:::lapply(housing_sub,function(x) paste(as.character(x$ST),"AAA",sep="_"))
take(aggreg,10);


# Example 2
df <- read.df(sqlContext, "./iris.data");
training <- filter(df, df$Species != "setosa");
model <- glm(Species ~ Sepal_Length + Sepal_Width, data = training, family = "binomial");
```


* Example 2 from Spark web-site

Barcelona Supercomputing Center
Centro Nacional de Supercomputación
EXCELENCIA SEVERO OCHOA

Python – SK-Learn + Theano: GPUs

```
from sklearn_theano.datasets import fetch_asirra
from sklearn_theano.feature_extraction import OverfeatTransformer
from sklearn_theano.utils import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report, accuracy_score

asirra = fetch_asirra(image_count=20)
X = asirra.images.astype('float32')
y = asirra.target
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.6, random_state=1999)
tf = OverfeatTransformer(output_layers=[-3])
clf = LogisticRegression()

pipe = make_pipeline(tf, clf)
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)

print(classification_report(y_test, y_pred))
print()
print("Accuracy score")
print("==============")
print(accuracy_score(y_test, y_pred))
```


* Example from scikit-learn.org : sklearn-theano

Weka and Moa – Examples

Weka & MOA (Using RMOA)

```
## HoeffdingTree example
require(RMOA);
hdt <- HoeffdingTree(numericEstimator = "GaussianNumericAttributeClassObserver");

## Define a stream - e.g. a stream based on a data.frame
data(iris);
iris <- factorise(iris);
irisdatastream <- datastream_dataframe(data=iris);

## Train the HoeffdingTree on the iris dataset
mymodel <- trainMOA(model = hdt, formula = Species ~ Sepal.Length + Sepal.Width +
Petal.Length, data = irisdatastream);

## Predict using the HoeffdingTree on the iris dataset
scores <- predict(mymodel, newdata=iris, type="response")
scores <- predict(mymodel, newdata=iris, type="votes")
```

* Example from BNOSAC: www.bnosac.be/index.php/blog/32-rmoa-massive-online-data-stream-classifications-with-r-a-moa

The classic MNIST (handwritten number recognition)

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

y = tf.nn.softmax(tf.matmul(x, W) + b)
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

* Example from tensorflow.org

ElasticSearch Example

ElasticSerarch queries

```
// Get documents with “mill” OR “lane” in address
{"query":{"match":{"address":"mill lane"}}

// Get documents with “mill lane” in address
{"query":{"match_phrase":{"address":"mill lane"}}}

// Get documents with “mill” AND “lane” in address
{"query":{"bool":{
     "must":[
        {"match":{"address":"mill”}},
        {"match":{"address":"lane”}}
      ]
}}}

// Requirement “must” (AND, ALL) can also be “should” (OR, AT LEAST ONE) or “must not” (NOR)

// Queries can be a composition
{"query":{"bool":{
      "must":[{"match":{"age":"40"}}],
      "must_not":[{"match":{"state":"ID"}}]
}}}
```

– Results are ranked according to the matching

* Examples from elastic.co : https://www.elastic.co/guide

SQL + R – Examples

SQL + R: User Defined Functions

- R code (analytics, machine learning, etc...) as SQL procedures
  - The code is executed as a Procedure the SQL Execution Tree
  - Inputs are considered DataFrames.
  - Outputs are considered Relations

Example:

```
## R code
PREDICTION <- function (x1,x2) { x1 * 10 + x2; }

-- SQL code
SELECT t.id AS ID, PREDICTION(t.a,t.b) AS
Prediction
FROM table AS t
WHERE t.c > 10;
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

CONCLUSIONS AND REMARKS

Conclusions / Remarks

- When benchmarking a system towards ML:
  - Several kinds of methods must be tested:
    - Spatial (e.g. k-NN), counting (e.g. Naïve Bayes), iterative (N-Nets)…
  - Different scenarios contemplated:
    - "Learning + slow new arrivals", "Fast Learning + Heavy new data", …
  - Methods can have different kind of inputs:
    - Numerical (easy to compute), Strings (hard to parse), Labels (easy to treat)
  - Preprocessing algorithms are also part of the ML process
    - If data requires binarization, string treatment to labels, etc… this phase is part of the ML process

- The same way a DB benchmark includes different queries, a DB-ML benchmark should:
  - Include different algorithms
  - Pointing towards different kinds of data
  - Focusing on learning, then on processing a high amount of data
  - Include the preprocessing as another algorithm

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Thanks for Your Attention

josep.berral@bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

AUXILIAR SLIDES – ML EXAMPLES

Quick example (part 1)

- Learning the Iris Data-Set*
  - Measures of flowers and its classification
    1. x_1: sepal length in cm
    2. x_2: sepal width in cm
    3. x_3: petal length in cm
    4. x_4: petal width in cm
    5. class: [Iris Setosa, Iris Versicolor, Iris Virginica]
  - We have labeled samples:
    - 5.1, 3.5, 1.4, 0.2, Iris-setosa
    - 7.0, 3.2, 4.7, 1.4, Iris-versicolor
    - 5.8, 2.7, 5.1, 1.9 ,Iris-virginica
    - ...
  - Given any flower, we want to know to which class it belongs
  - We want to find a function: $f(x_1,x_2,x_3,x_4) \rightarrow$ class

*R.A. Fisher (1936). Source: https://archive.ics.uci.edu/ml/datasets/Iris

Quick example (part 2)

- Learning the Iris Data-Set*
  - Learning it on a decision tree (C4.5)

1. Decide the splits: 200 instances = 50% training, 25% validation, 25% test
2. Decide algorithm parameters (minimal samples per branch = 2, …)
3. Train the model and test against validation

```
Correctly Classified Instances          49               96.0784 %
Incorrectly Classified Instances         2                3.9216 %

=== Confusion Matrix ===

  a  b  c   <-- classified as
 15  0  0 |  a = Iris-setosa
  0 19  0 |  b = Iris-versicolor
  0  2 15 |  c = Iris-virginica
```

4. Iterate 2-3 changing parameters until the model satisfies us
5. Apply the test data-set and get final evaluation for our model

*R.A. Fisher (1936). Source: https://archive.ics.uci.edu/ml/datasets/Iris

Quick example (part 3)

- Learning the CPU Data-Set*
  - Properties of a CPU and its relative performance
    1. x_1: vendor name: 30 (adviser, amdahl ,apollo, basf, bti, burroughs, …)
    2. x_2: Model Name: many unique symbols
    3. x_3: MYCT: machine cycle time in nanoseconds (integer)
    4. x_4: MMIN: minimum main memory in kilobytes (integer)
    5. x_5: MMAX: maximum main memory in kilobytes (integer)
    6. x_6: CACH: cache memory in kilobytes (integer)
    7. x_7: CHMIN: minimum channels in units (integer)
    8. x_8: CHMAX: maximum channels in units (integer)
    9. x_9: PRP: published relative performance (integer)
    10. x_{10}: ERP: estimated relative performance from measures (integer)
  - We have labeled samples:
    - 125, 256, 6000, 256, 16, 128, 198
    - 29, 8000, 32000, 32, 8, 32, 269
    - 29, 8000, 32000, 32, 8, 32, 220
    - ...
  - Given any CPU, we want to know its estimated performance
  - We want to find a function: $f(x_1,x_2,\dots,x_9) \rightarrow$ value

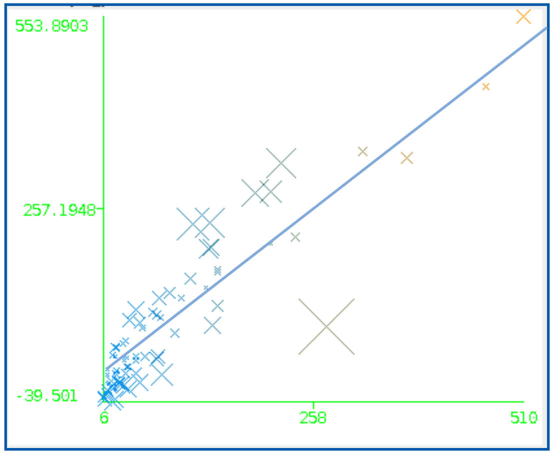
* P.Ein-Dor and J. Feldmesser Source: https://archive.ics.uci.edu/ml/datasets/Computer+Hardware

Quick example (part 4)

- Learning the CPU Data-Set*
  - Learning it on a Linear Regression

1. Decide the splits: 200 instances = 50% training, 25% validation, 25% test
2. Decide algorithm parameters (ridge = $1e10^{-8}$, …)
3. Train the model and test against validation

```
Correlation coefficient                    0.9158
Mean absolute error                       38.1617
Root mean squared error                   48.9672
Relative absolute error                   45.5102 %
Root relative squared error               46.332  %
Total Number of Instances                 71
```



4. Iterate 2-3 changing parameters until the model satisfies us
5. Apply the test data-set and get final evaluation for our model

* P.Ein-Dor and J. Feldmesser Source: https://archive.ics.uci.edu/ml/datasets/Computer+Hardware